A Repository System with Secure File Access for Collaborative Environments ⋆

Paul A. Gray†, Srividya Chandramohan‡ and Vaidy S. Sunderam‡

†Dept. of Computer Science
University of Northern Iowa
Cedar Falls, Iowa
50614-0506
gray@cns.uni.edu

‡Dept of Math & Computer Science
Emory University
Atlanta, Georgia
30332
{schand2 | vss}@emory.edu

Abstract. Collaborative computing environments which allow remote execution of applications, need a remote storage facility that supports shared access to the software resources required for computation. In this context, there is also a need to guarantee authorized and secure access to the shared resources. This paper investigates the use of a repository system in collaborative computing environments and discusses techniques to provide privacy, user authentication and access control to the repository using certificates. A protocol based on SSL is developed for query processing. The IceT environment is used as an exemplar for the application of the secure repository system.

1 Introduction

In recent years, the use of network environments as a platform for high performance distributed computing, has become popular. Research collaborations are being formed by merging geographically-distributed environments [5]. These collaborations often pool resources together in order to tackle a common goal.

Each member in any of the groups can contribute software and data to be shared by other members of the group. There are several means to share data across networks varying from simple file transfer to complex distributed database management and distributed shared memory approach. These methods for file sharing lack the ability to support a dynamically changing membership of users within the groups. They also lack mechanisms to authenticate users in dynamic environments.

One model is to have applications and data stores in a repository to facilitate group access. Due to the dynamic nature of the collaborative resource alliance, the repository system cannot be managed with physical user accounts on the machines that host the repositories. In this paper, we present a simple model of a repository system that suits the computing requirements of dynamic collaborative environments. Section 2.2 outlines the design and discusses the usefulness of this repository system.

When an alliance is formed and data from the repository has to be shared, issues such as secure communication, authentication and authorization have to be tackled. The Grid Security Infrastructure (GSI) [9], developed within the Globus project [8] addresses these issues in detail and defines a security policy by mapping interdomain

⋆ Research supported by NSF grant ACI-9872167 and the University of Northern Iowa's Graduate College.

V.N. Alexandrov et al. (Eds.): ICCS 2001, LNCS 2073, pp. 404–412, 2001.
Springer-Verlag Berlin Heidelberg 2001

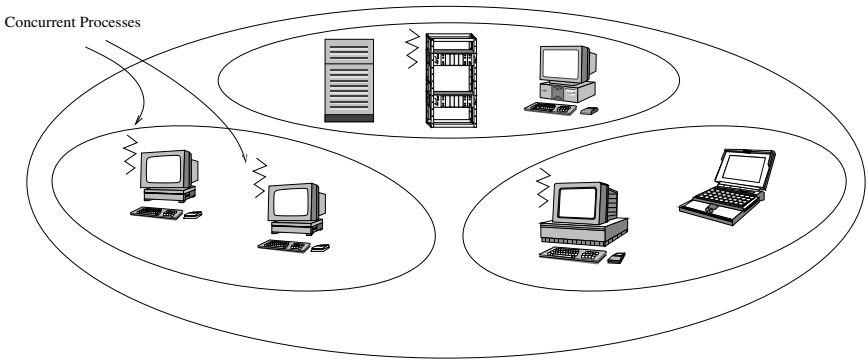

Fig. 1. Dynamic Collaborative Environment.

operations to local security policy. Further, Akenti [16] implements an automated access control mechanism using digitally signed certificates.

Lack of a local security policy or local database administration and the dynamic nature of the collaboration necessitate development of an access control mechanism that is more dynamic and adaptable than the GSI and Akenti infrastructures. The relevant security policies and associated protocols for query processing used in this model are described in Section 3. We also present a reference implementation in the context of IceT specifically [17], and applicable to the repository frameworks in general.

2 The Repository System

In this section we discuss the background and need for a repository system. The details of the proposed model are also outlined.

2.1 Background and Need

The Internet enables users to access resources and run applications over a heterogeneous collection of computers and networks. Figure 1 shows an example of a dynamic environment in which three heterogeneous, networked environments have been merged to form a single virtual machine. Processes can now migrate across environments for optimal resource utilization.

In these and other scenarios, when code is sent from one computer to another and run at the destination, problems such as portability of machine code, commanality of data representation formats and data conversion for network transfer arise. For example, PC users sometimes send executable files as email attachments to be run by the recipient, but a recipient will not be able to run it, for example, on a Macintosh computer. These problems are overcome using 1) the *virtual machine* approach, such as in Java, as a way of making code executable on any hardware, and 2) *External data representation* as an agreed standard for the representation of data. Sun XDR, CORBA's common data representation (CDR) and Java's object serialization are examples of the latter.

Given the use of a suitable scheme to mask the heterogeneity in distributed systems, this paper looks at some specific computing requirements that are not effectively met by current systems and provides a solution for the same.

Consider the case where Java programs are used in a collaborative environment. These programs may access legacy codes written in C, C++ or Fortran for high performance computations. Let Bob and Alice be two end communicating parties (or machines). Bob sends program called "foo" to Alice via a communication link. To run "foo", Alice has to: locate and resolve static dependencies (java classes and methods), locate and link external shared libraries (e.g. a DLL for a windows machine) and provide for external user data that the program might require for processing.

These dependencies can be easily resolved if Alice can locate the files on her local filesystem. But Alice has no prior knowledge of the resources required and need not maintain a reserve. Alice could interactively request Bob to send these files and Bob can respond back with the required files. If Bob sends the program to Sue as well, with Sue working on another machine within the same network as that of Alice, Bob has to duplicate the file transfer. Suppose that Bob wants to do some statistical analysis and needs Carol and Dave on a different network to participate in the computation concurrently, Bob has to maintain persistent connection with all four parties and service their needs. This will be a potential performance bottleneck. Alternately, in an agent-based scenario, Bob could generate an agent process on Alice's machine. Then Bob's machine is not needed again even if the process migrates to Carol's system.

Irrespective of the programming paradigm used, such problems persist in collaborative computing environments. *There is no Network File System for such environments* that facilitates shared access to applications, libraries and user data for all members in the communion, catering to the different underlying operating systems and architectures. In this vein, we propose a simple *repository system* for dynamic environments, supporting mobile agents (by serving up files tailored to operating system/architecture) and remote data storage. Continuing with the example scenario, Bob can now store the Java files with architecture and operating system-specific libraries in a repository and authorize Alice, Sue, Carol and Dave to access the repository. When the group is formed, the parties are given the knowledge of the available repositories so that they can fetch data on-demand.

2.2 Details of the Model

The scope of the repository system is to address the computing needs of dynamic environments in which processes participate in remote execution and remote access of resources within the virtual machine environment. Java applications that use legacy codes for high performance computations are targetted. The repository system fills in the need for a shared filesystem. In this model, which epitomizes the breadth of the implementation, users store Java-based front-ends and supporting native library formats for different architectures that might be called upon to run processes. Figure 2 shows possible contents of a repository.

The model also supports the following features:

- Repositories are not restricted to locations within the virtual machine. Bob can create a repository on his machine and so can Alice. Bob and Alice could be within the same intranet or otherwise.
- Users can add to or delete from, a local or remote repository based on a global access policy.
- The owner of a resource can impose security restrictions for availability and accessibility of the resource. There is no central authority to impose access restrictions on all the resources.

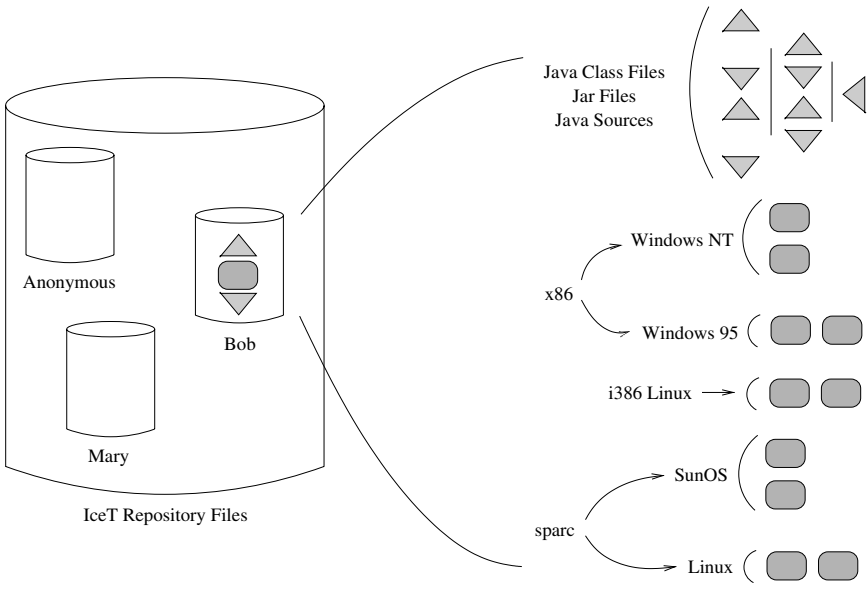

Fig. 2. Repository Files Belonging to Users.

The dynamic nature of the collaboration and the lack of central authorities, hinder the application of conventional distributed database management models where database access is administered by setting up user accounts and passwords. There is a need to specify access control policies that are non-account based and do not rely upon the presence of a database administrator. The access control policy for the repository system is described in Section 3.

We adopt a User Interface Distribution model (see [12]) in the three-tier client/server architecture. The layers are represented by *hosts*, *clients* and *servers*. Figure 3 illustrates the model. Host performs data-access processing i.e accesses data from the disk. Client performs user interface processing. It contains GUI interfaces and additional rules such as client certificates (for authentication). Server performs function processing. It stores constraints that are used to access data from the host.

Instead of accessing the repository via a standard interface (such as JDBC or ODBC), the client sends queries to the server and the server processes the requests. Thus the server acts as a conduit for passing processed data from host to client. The viablity of this approach stems from two reasons: (1) the client can run in a computing system different from the host and the server, and (2) access control mechanisms can be implemented independent of the type of interface to the repository system.

3 Security, Authorization and Access Control

The centralized storage of data in databases, and the accessing of this data by multiple end users, bring with them a need for security. The model must have mechanisms that will allow users to access data they need yet prevent them from accessing data they are not authorized to see. In addition to controlling the data a particular user has access to,

the repository system should control the type of access the user has, such as whether the user is allowed only to retrieve the data or may also make changes to it or add new data to the database.

As noted in Section 2.2, due to the dynamic characteristics of the alliance, where new users can join and existing users can leave the group at anytime, users do not have accounts and passwords on the repository host. Hence user authentication mechanisms which depend on account setup such as Kerberos [13] and SSH [11] cannot be used. This also eliminates risks due to "password sniffing" [7].

Instead, Certificates are used both for authorizing access to the repository and for governing subsequent access to files. The group has to maintain Certificate Authorities (CA) which issue and sign certificates for the group members. The repository server stores this "trust" information. When Bob inserts his files into the repository, he presents to the repository server, a valid certificate digitally signed by a CA that the server trusts. This authorizes Bob to access the repository. Bob could grant access privileges to Alice by signing Alice's certificate already signed by a group CA. This creates a certificate chain of users. Along with the files owned by Bob, access permissions in the form of user certificates will be stored in the repository. Bob could also specify some of his files to be public so that anonymous users can read those files. The server has to check the permissions before granting access. When Bob changes his mind, and wants to revoke privileges granted for Alice, he will revoke Alice's certificate. A Certificate Revocation List (CRL) is maintained by the group authority for each user. Alternately, the resource alliance can be terminated and all access privileges are revoked for all users. In both cases, *a consistent state of the repository on the host has to be maintained*. The access control list for a user in the repository has a "checkCRL" flag. Revocation of privileges is reflected by setting the flag, so that the server can query the user's CRL prior to servicing any file requests by the user.

When a user posts requests, the repository server's reply may constitute a file transfer. Data is typically sent via an insecure communication channel. To ensure privacy , the data has to be encrypted. This issue has been addressed in depth in projects such

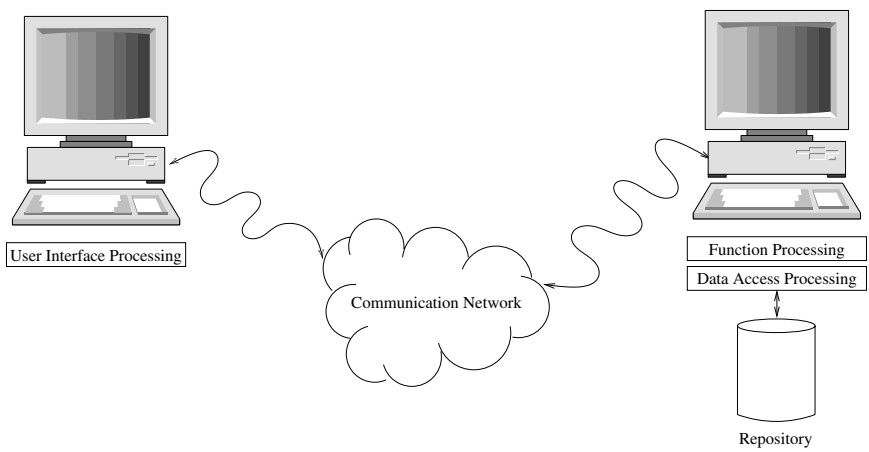

Fig. 3. User Interface Distribution Model.

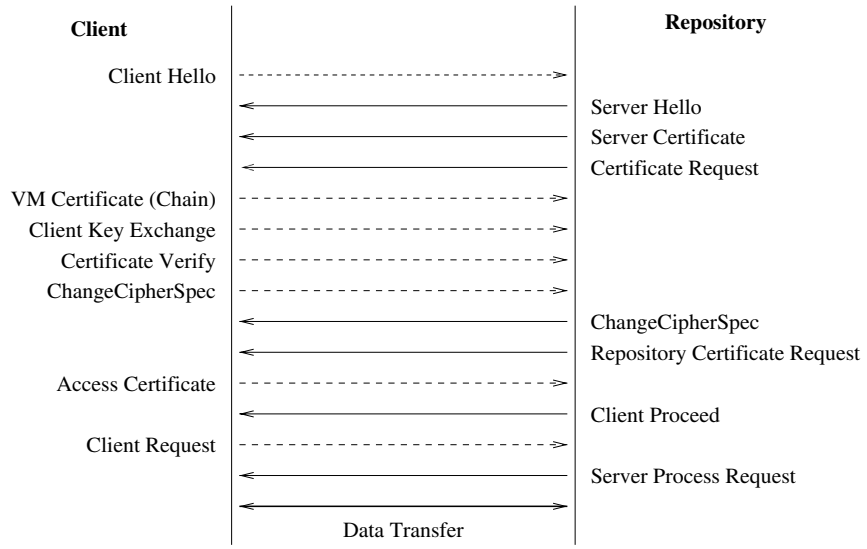

Fig. 4. The Protocol Established For Accessing Files In A Repository.

as Globus [8], Akenti [16] and securePVM [18]. The channel should also be protected from network attacks such as eavesdropping, masquerading, message tampering, replaying, denial of service etc. (see [14]). This calls for making the channel secure. The SSL protocol [4] can be used to establish a secure channel. In an open network, all client parties may not use the same client software. The client and server software may not support same encryption algorithms. SSL is a good choice because it is designed so that the algorithms used for encryption and authentication are negotiated between the processes at the two ends of the connection during the initial handshake.

3.1 Secure Query Processing Protocol

The protocol used to establish contact with the repository is similar to SSL, with modifications to allow the use of certificate chains belonging to users in the merged environments. Figure 4 depicts the protocol.

The protocol messages are sent in the following order:

1. Client hello - The client sends the repository server information including the highest version of SSL it supports and a list of the cipher suites it supports. The cipher suite information includes cryptographic algorithms and key sizes.
2. Server hello - The server chooses the highest version of SSL and the best cipher suite that both the client and server support and sends this information to the client.
3. Server Certificate - The repository server sends the client a certificate or a certificate chain. This message is used to authenticate the repository server.
4. Certificate request - The server then issues this message to the client, which contains a list of acceptable Distinguished Names (DN) that are recognized as credible.
5. VM Certificate (Chain)- The client sends its certificate (chain), just as the server did in Message 3. The client has to send a certificate that has been certified by one of the listed DNs.

6. Client key exchange - The client generates information used to create a key to use for symmetric encryption.
7. Certificate verify - The client sends information that it digitally signs using a cryptographic hash function. When the server decrypts this information with the client's public key, the server is able to authenticate the client.
8. Change cipher spec - The client sends a message telling the server to change to encrypted mode.
9. Change cipher spec - If the server accepts the certificate as valid, a response to the change in cipher is issued.
10. Repository Certificate Request - At this point, the client has authenticated itself to the server and a secure channel has been established. However, another valid certificate is required to access files on the repository as the owner of the files may have imposed access restrictions. The server requests the client for the certificate corresponding to the owner of the repository files.
11. Access Certificate - The client's certificate signed by the owner is presented if one is available, else a NoCertificateAlert message is sent.
12. Client Proceed - If the server can verify the certificate presented, it grants access to the client. Otherwise the access is denied and the session is closed. It allows anonymous access if it received a NoCertificateAlert in message 11.
13. Client Request - The client requests to access or update files in the repository and sends the details (file name, architecture, OS) of the file it requests.
14. Data Transfer - The server queries the access control list to check if the particular file can be accessed. If the checkCRL flag is set, the client's CRL is checked for validity before granting access to the file.

4 A Reference Implementation of the Repository System

In this section, we describe the IceT repository system, an implementation of our proposed model. The repository system was developed as a part of IceT project [17], whose focus is to:

- build distributed applications using multiple heterogeneous environments.
- support the use of portable shared libraries.
- target applications to dynamic reconfigurable environment which allows merging and splitting of environments.
- address security concerns
- provide an environment suitable for collaboration and distributed computing applications

We describe here selected aspects of this implementation, focusing on our use of the Java Secure Socket Extension (JSSE) API for SSL, a Postgres database and Java Keytool for certificates.

4.1 Use of Java Secure Socket Extension API and Keytool

The Java Secure Socket Extension (JSSE) [1] provides a framework and a reference implementation for a Java version of the SSL protocol and includes functionality for data encryption, server authentication, message integrity, and client authentication. The JSSE API is used for creating and configuring secure socket factories. The Java Keytool is used to generate keys (inserted into a keystore), certificate signing requests and

to import trusted X509 certificates into a user defined truststore. For details of using x.509 certificates see [10]. For testing purposes, OpenSSL `ca` command (as in [2]) was used to sign the certificate requests and create a chain of trusted of X509 certificates. The keystore and truststore are loaded into X509 key and trust managers respectively. The API also provides a class representing a secure socket protocol implementation. A Query Processing protocol as described in Section 3.1 is implemented on top of SSL to access the repository. These tools come together in our implementation to facilitate dynamic, short-term collaborative alliances,where self-signed certificates and CRLs are used on a more intimate framework of users. I.e. users will hold self-signed certificates and act as their own certificate authorities.

4.2 Postgres Database

PostgreSQL [3] was our implementation choice for the database. Postgres uses a simple "process per-user" client/server model. A Postgres session consists of a supervisory daemon process, the user's frontend application (eg. psql program) and one or more backend database servers. A single postmaster manages the repositories on the host. The postmaster is always running, waiting for requests, whereas frontend and backend processes come and go.

A primary registry table is created to store X509 certificates of different users and their corresponding *user-id*. A table indexed by *file-id* stores Java source files for each user and a library table stores the user's libraries based on architecture and operating system. A separate access permissions table maintains access list for files belonging to a user. Any query posted to the repository has to be processed as:

$$\text{user certificate} \rightarrow \text{user-id} \rightarrow \text{file-id} \rightarrow \text{file permissions} \rightarrow \text{file contents}$$

I.e., each access to the repository requires a valid X.509 certificate and access permissions are checked prior to granting file access.

Once a secure channel is established using JSSE methods, the user sends his queries to the server. The server connects to the postgres host (on same machine as the server) via the JDBC interface. The user certificate is matched to that in the repository, access permissions are checked and the encrypted file is transferred to the requesting client.

5 Conclusions and Future Work

We have described a model for distributed access of resources with security and authorization features. The model is general enough to be suited to several applications including but not limited to:

- collaborative computing projects such as Harness [15] and CCF [6]
- security policy for remote execution and remote access of resources
- extending security architecture given by Globus and Akenti for dynamic environments
- access control policy for distributed databases using certificates

The prototype implementation has shown the feasibility of the certificate-based authentication and secure repository model. Preliminary benchmarks have also shown that there can be considerable overhead associated with SSL channel initialization and encryption of the data stream. Our current design does not clearly specify interactions

between the repository system and the distributed applications in which it is used. It does not yet address functions pertinent to distributed databases such as transparency control, concurrency etc. Some of the improvements in the security policy include providing for validation of files within the repository using message digests and encrypting data stored in the repository. Our future work also includes implementing the repository system using a language-independent scheme so that it can be easily adapted to existing systems.

References

1. URL: http://java.sun.com/products/jsse.
2. URL: http://www.openssl.org/doc/.apps/openssl.html.
3. PostgreSQL user's guide. URL: http://www.postgresql.org/docs/user/user.html.
4. A. O. Freier, P. Karlton and P. C. Kosher. *The SSL Protocol, version 3.0.* Netscape Communications, Internet Draft, Nov 1996. URL: http://www.netscape.com/eng/ssl3/.
5. C. Catlett and L. Smarr. Metacomputing. *Communications of The ACM*, 35(6):44–52, 1992.
6. S. Chodrow, S. Cheung, P. Hutto, A. Krantz, P. Gray, T. Goddard, I. Rhee, and V. Sunderam. CCF: A Collaborative Computing Frameworks. In *IEEE Internet Computing*, Jan/Feb 2000.
7. Computer Emergency Response Team. Ongoing Network Monitoring Attacks. CERT Advisory: CA - 94:01, Feb 1994.
8. I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputing Applications*, May 1997.
9. I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *ACM Conference on Computers and Security*, pages 83–91. ACM Press, 1998.
10. Internation Telecommunication Union. X.509: Information Technology - open systems interconnection - the directory: Public-key and attribute certificate frameworks. ITU-T Recommendation, Mar 2000. To be published.
11. J. Barrett and R. Silverman. *SSH, The Secure Shell: The Definitive Guide*. O'Reilly, 1st edition, 2001.
12. J. Martin and J. Leben. *Client/Server Databases - Enterprise Computing.* Prentice Hall P T R, 1995.
13. J. Steiner and C. Neuman and J. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *Usenix Conference Proceedings*, 1988.
14. M. Dekker. Security of the Internet. *The Froehlich/Kent Encyclopedia of Telecommunications*, 15:231–255, 1997. URL: www.cert.org/encyc_article/tocencyc.html.
15. M. Migliardi and V. Sunderam. Heterogeneous Distributed Virtual Machines in the Harness Metacomputing Framework.
16. M. Thomson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, and A. Essari. Certificate-based access control for widely distributed resources. In *Proceedings of the Eighth Usenix Security Symposium*, Aug 99.
17. P. Gray and V. Sunderam. IceT: Distributed Computing and Java. *Concurrency: Practice and Experience*, 11(9):1161–1167, Nov 1997.
18. T. H. Dunigan and N. Venugopal. Secure PVM. Technical Report TM-13203, Oak Ridge National Laboratories, Aug 1996.